YEI Skeleton API Quick Start Guide

Purpose

The purpose of this document is to help users integrate the YEI Skeleton Application Programming Interface (API) into a Visual Studio project. After reading this document, users will be able to integrate the API into various projects and interact with the internal skeleton.

Overview

The YEI Skeleton API is a collection of convenience functions for creating and using skeletons with YEI Prio and 3-Space Sensor devices or any other kind of device for use in a program written in C/C++ or any language that can import a compiled library (.dll, .so, etc).

Software Requirements

Operating System – Windows 7 32-Bit/64-Bit
(*Windows XP 32-Bit/64-Bit*, *Windows Vista 32-Bit/64-Bit*, *Windows 8*, and *Unix* are untested)

Getting Started with the Skeleton API

When creating and communicating with an internal skeleton via the API, there are a few common steps that must be taken before full communication may take place. These steps are as follows:

1. Link to the API library.
2. Include the API header in your file.
3. Create a skeleton.

If guidance is required for linking the library to the user's development environment, please refer to one of the following Quick Starts:

YEI Skeleton API Quick Start Guide: Setup Environment for Visual Studio

The next section will go over the last step using code snippets from the example *creating_instances.c* and explain some key components in using the API. For a more detailed description of the API and its components, please refer to the *YEI Skeleton API User's Manual* or the documentation file for the YEI Skeleton API.

Creating a Basic C Code Example

This example and more like it can be found on the YEI website (www.yeitechnology.com).

Creating Instances

This example shows how to use the API to create an internal skeleton. When a skeleton is created a *yei_skeleton_id* is returned, which is needed to call many functions in the API.

```
#include <stdio.h>
#include <stdlib.h>
#include "yei_skeleton_api.h"

int main()
{
    yei_skeleton_id skel_id;
```

The API has several methods to automate skeleton creation for the user. One such method generates a skeleton based on sex and age of the user.

```
    uint8_t is_male = 1;    // <-- Edit this to your user's sex
    uint32_t age = 26;      // <-- Edit this to your user's age
    printf("Creating Skeleton for a %s, %d Years Old\n", is_male ? "Male" : "Female", age);
    skel_id = yeiskel_createStandardSkeletonWithAge(is_male, age);
```

When a skeleton is created it is always best to check if the *yei_skeleton_id* is valid. If the *yei_skeleton_id* is valid users can then call methods for the skeleton. As this example just covers creating skeletons we will go ahead and destroy it.

```
    if (skel_id == YEI_SKELETON_INVALID_ID)
    {
        printf("Failed to create a skeleton for a %s, %d years old\n",
                                                       is_male ? "Male" : "Female", age);
    }
    else
    {
        printf("Created a skeleton for a %s, %d years old\n", is_male ? "Male" : "Female", age);
        yeiskel_destroySkeleton(skel_id);
    }
    printf("================================\n");
```

Another method of generating a skeleton is to using the user's sex and height (in meters).

```
    float height = 1.8f;    // <-- Edit this to your user's height
    printf("Creating Skeleton for a %s, %f Meters Tall\n", is_male ? "Male" : "Female", height);
    skel_id = yeiskel_createStandardSkeletonWithHeight(is_male, height);
    if (skel_id == YEI_SKELETON_INVALID_ID)
    {
        printf("Failed to create a skeleton for a %s, %d years old\n",
                                                       is_male ? "Male" : "Female", age);
    }
    else
    {
        printf("Created a skeleton for a %s, %d years old\n", is_male ? "Male" : "Female", age);
        yeiskel_destroySkeleton(skel_id);
    }
    printf("================================\n");
```

If a user prefers not to use a generated skeleton the API also allows users to load in skeletons hierarchies in XML format.

```
    const char* xml_file = "./YEI_Skeleton_Hierarchy.xml";  // <-- Edit to user's profile file
    printf("====Creating Skeleton from File====\n");
    skel_id = yeiskel_createSkeletonFromFile(xml_file);
    if (skel_id == YEI_SKELETON_INVALID_ID)
    {
        printf("Failed to create a skeleton for a %s, %d years old\n",
                                                     is_male ? "Male" : "Female", age);
    }
    else
    {
        printf("Created a skeleton for a %s, %d years old\n", is_male ? "Male" : "Female", age);
        yeiskel_destroySkeleton(skel_id);
    }
    printf("================================\n");
```

Once finished with the YEI Skeleton API and before quitting from your code, it is best to call the *yeiskel_resetSkeletonApi* function to properly clean-up the memory the API has allocated. This also helps assure the API will work properly the next time it is used. The *yeiskel_resetSkeletonApi* function as with many other YEI Skeleton API functions will return a *YEI_SKELETON_ERROR*. Checking this error to see if the command succeeded is always best. This will help when debugging to see what went wrong.

```
    YEI_SKELETON_ERROR error = yeiskel_resetSkeletalApi();
    if (error != YEI_SKELETON_NO_ERROR)
    {
        printf("ERROR: %s\n", yei_skeleton_error_string[error]);
    }
    return 0;

    printf("\nFinished press Enter to continue");
    getchar();
    return 0;
}
```

Getting Data

This example shows how to use the API to get information from the internal skeleton. First a skeleton must be generated.

```
#include <stdio.h>
#include <stdlib.h>
#include "yei_skeleton_api.h"
int main()
{
    yei_skeleton_id skel_id;
    uint8_t is_male = 0;    // <-- Edit this to your user's sex
    uint32_t age = 24;      // <-- Edit this to your user's age
    printf("Creating Skeleton for a %s, %d Years Old\n", is_male ? "Male" : "Female", age);
    skel_id = yeiskel_createStandardSkeletonWithAge(is_male, age);
    if (skel_id == YEI_SKELETON_INVALID_ID)
    {
        printf("Failed to create a skeleton for a %s, %d years old\n",
                                                     is_male ? "Male" : "Female", age);
    }
```

Once the skeleton has been created we will need to get names of the bones we want to get information from. Using *yeiskel_getStandardBoneName* we can retrieve the YEI standard bone name for the head bone. When using an alternate naming convention the YEI Skeleton API allows for alias naming of bones.

```
        char head_bone[32];
        YEI_SKELETON_ERROR error;
        error = yeiskel_getStandardBoneName(YEI_SKELETON_STANDARD_BONE_HEAD, head_bone, 32);

        error = yeiskel_addBoneAlias(skel_id, "Head", head_bone);
```

Using the bone or alias name we may now find out information about the specified bone.

```
        float quat[4];
        float vel[3];
        float pos[3];

        error = yeiskel_getBoneOrientation(skel_id, head_bone, quat);
        if (error == YEI_SKELETON_NO_ERROR)
        {
            printf("Quat: %f, %f, %f, %f\n", quat[0], quat[1], quat[2], quat[3]);
        }

        error = yeiskel_getBoneVelocity(skel_id, head_bone, vel);
        if (error == YEI_SKELETON_NO_ERROR)
        {
            printf("Vel: %f, %f, %f\n", vel[0], vel[1], vel[2]);
        }
        error = yeiskel_getBonePosition(skel_id, head_bone, pos);
        if (error == YEI_SKELETON_NO_ERROR)
        {
            printf("Pos: %f, %f, %f\n", pos[0], pos[1], pos[2]);
        }
```

Once finished gathering data from the skeleton remember to destroy the skeleton and reset the API. Doing so will properly clean-up the memory the API has allocated.

```
        yeiskel_destroySkeleton(skel_id);
        error = yeiskel_resetSkeletalApi();
        if (error != YEI_SKELETON_NO_ERROR)
        {
            printf("ERROR: %s\n", yei_skeleton_error_string[error]);
        }
        return 0;

        printf("\nFinished press Enter to continue");
        getchar();
        return 0;
    }
```

Using Skeleton Processors

A skeleton processor is an instance that runs a process on a single or many internal skeleton instances. The YEI Skeleton API has several built-in processors along with a base class that can be inherited from to create custom skeleton processors. This example will go over setting up built in *PrioConnection* processor and using it with a Prio Lite suit. As in previous examples first the skeleton must be created.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <stdint.h>
#include "yei_skeleton_api.h"

int main(){
    yei_skeleton_id skel_id;

    uint8_t is_male = 0;    // <-- Edit this to your user's sex
    uint32_t age = 34;      // <-- Edit this to your user's age
    printf("Creating Skeleton for a %s, %d Years Old\n", is_male ? "Male" : "Female", age);
    skel_id = yeiskel_createStandardSkeletonWithAge(is_male, age);
    if (skel_id == YEI_SKELETON_INVALID_ID)
    {
        printf("Failed to create a skeleton for a %s, %d years old\n",
                                                  is_male ? "Male" : "Female", age);
    }
    printf("================================\n");
```

After the skeleton is created we will set the pose of the skeleton to the standard “T” pose. This will allow the user of the Prio suit to know the starting pose.

```
    printf("====Setting a Pose for Skeleton====\n");
    YEI_SKELETON_ERROR error = yeiskel_setSkeletonToStandardTPose(skel_id);
    if (error != YEI_SKELETON_NO_ERROR)
    {
        printf("ERROR: %s\n", yei_skeleton_error_string[error]);
    }
    printf("================================\n");
```

Once the skeleton is set up we will create the *PrioConnection* processor and add it to the skeleton's list of processors. By calling *yeiskel_createPrioProcessor* the API will automatically establish a connection with the Prio Base Station based on the com_offset parameter, then enumerate and set up the paired Prio Hub. The return value will be a *yei_skeleton_id* that we be used to call functions on the processor.

```
    printf("====Creating a Prio Processor====\n");
    yei_skeleton_id prio_id = yeiskel_createPrioProcessor(0);
    if (prio_id == YEI_SKELETON_INVALID_ID)
    {
        printf("Failed to create Prio Processor\n");
    }
    printf("================================\n");
```

Adding a processor to the skeleton allows the processor to manipulate the bones in the skeleton. When adding the processor to the skeleton there is an index parameter that tells the skeleton what priority the processor should have when running. The priority runs in ascending order with zero having the highest priority. For convenience users can use *YEI_SKELETON_PROCESSOR_LIST_END* to ensure lowest priority.

```
        printf("====Adding the Prio Processor to the Skeleton====\n");
        error = yeiskel_addProcessorToSkeleton(skel_id, 0, prio_id);
        if (error != YEI_SKELETON_NO_ERROR)
        {
            printf("ERROR: %s\n", yei_skeleton_error_string[error]);
        }
        printf("================================\n");
```

Alternatively to adding bones manually to the skeleton the YEI Skeleton API can build skeletons by reading skeleton XML files. The API includes several methods to generate XML strings to match the standard Prio Lite, Core and Pro layouts. Calling *yeiskel_setStandardDeviceXmlMapPrioLiteLayout* will set up the skeleton's bones to match the standard Prio Lite layout.

```
        printf("====Mapping the Skeleton to Match the Prio Lite Suit====\n");
        error = yeiskel_setStandardDeviceXmlMapPrioLiteLayout(skel_id);
        if (error != YEI_SKELETON_NO_ERROR)
        {
            printf("ERROR: %s\n", yei_skeleton_error_string[error]);
        }
        printf("================================\n");
```

Now that the skeleton is set up and linked to the processor we need to gather the bone names to retrieve the information. Placing the needed bone names into an array allows us to easily loop over all the bones to get information about each one.

```
        printf("====Setting up skeleton bones====\n");
        char name_buff[256];
        error = yeiskel_getRootBoneName(skel_id, name_buff, 256);
        if (error != YEI_SKELETON_NO_ERROR)
        {
            printf("ERROR: %s\n", yei_skeleton_error_string[error]);
        }
        printf("Getting standard bone names for Prio Lite suit\n");
        char names[9][32];
        yeiskel_getStandardBoneName(YEI_SKELETON_STANDARD_BONE_HIPS, names[0], 32);
        yeiskel_getStandardBoneName(YEI_SKELETON_STANDARD_BONE_SPINE, names[1], 32);
        yeiskel_getStandardBoneName(YEI_SKELETON_STANDARD_BONE_HEAD, names[2], 32);
        yeiskel_getStandardBoneName(YEI_SKELETON_STANDARD_BONE_LEFT_UPPER_ARM, names[3], 32);
        yeiskel_getStandardBoneName(YEI_SKELETON_STANDARD_BONE_LEFT_LOWER_ARM, names[4], 32);
        yeiskel_getStandardBoneName(YEI_SKELETON_STANDARD_BONE_LEFT_HAND, names[5], 32);
        yeiskel_getStandardBoneName(YEI_SKELETON_STANDARD_BONE_RIGHT_UPPER_ARM, names[6], 32);
        yeiskel_getStandardBoneName(YEI_SKELETON_STANDARD_BONE_RIGHT_LOWER_ARM, names[7], 32);
        yeiskel_getStandardBoneName(YEI_SKELETON_STANDARD_BONE_RIGHT_HAND, names[8], 32);
        printf("================================\n");
```

With the id for the *PrioConnection* processor we can now make calls to the API and set up the connected Prio device for streaming.

```
    printf("====Calibrating the Prio Suit====\n");
    error = yeiskel_calibratePrioProcessor(prio_id, 0.0f);
    if (error != YEI_SKELETON_NO_ERROR)
    {
        printf("ERROR: %s\n", yei_skeleton_error_string[error]);
    }
    printf("================================\n");

    printf("====Setting up the Standard Prio Streaming====\n");
    error = yeiskel_setStandardStreamPrioProcessor(prio_id);
    if (error != YEI_SKELETON_NO_ERROR)
    {
        printf("ERROR: %s\n", yei_skeleton_error_string[error]);
    }
    printf("================================\n");
```

Now that everything is set up we can start streaming to the processor with *yeiskel_startPrioProcessor.* To get data on the needed bones we will simply set up a loop that runs for 10 seconds.

```
    printf("====Start Streaming====\n");
    error = yeiskel_startPrioProcessor(prio_id);
    if (error == YEI_SKELETON_NO_ERROR)
    {
        printf("Streaming has started!\n");
        printf("================================\n");

        float quat[9][4];
        const float RUN_TIME = 10.0f;    // in seconds
        uint8_t i;
        clock_t start_time = clock();
        printf("\n====Getting Data from Skeleton====\n");
        while ((((float)(clock() - start_time)) / CLOCKS_PER_SEC < RUN_TIME))
        {
            error = yeiskel_update(skel_id);
            if (error == YEI_SKELETON_NO_ERROR)
            {
                for (i = 0; i < 9; i++)
                {
                    printf("====Bone: %s====\n", names[i]);
                    error = yeiskel_getBoneOrientation(skel_id, names[i], quat[i]);
                    if (error == YEI_SKELETON_NO_ERROR)
                    {
                        printf("Quat: %f, %f, %f, %f\n", quat[i][0], quat[i][1],
                                                         quat[i][2], quat[i][3]);
                    }
                }
            }
        }
        printf("================================\n");
```

When finished retrieving data we can stop the *PrioConnection* processor from streaming and clean-up any used memory by the API.

```
            printf("====Stop Streaming====\n");
            error = yeiskel_stopPrioProcessor(prio_id);
            if (error != YEI_SKELETON_NO_ERROR)
            {
                printf("ERROR: %s\n", yei_skeleton_error_string[error]);
            }
        }
        else
        {
            printf("ERROR: %s\n", yei_skeleton_error_string[error]);
        }
        printf("================================\n");

        yeiskel_destroySkeleton(skel_id);

        printf("====Resetting API====\n");
        error = yeiskel_resetSkeletalApi();
        if (error != YEI_SKELETON_NO_ERROR)
        {
            printf("ERROR: %s\n", yei_skeleton_error_string[error]);
        }

        printf("\nFinished press Enter to continue");
        getchar();
        return 0;
```

Now that you are more familiar with the functionality of the YEI Skeleton API and how to connect to and use skeleton processors, you can begin creating your own examples and applications using the API.

YEI Technology
630 Second Street
Portsmouth, Ohio 45662

Toll-Free: 888-395-9029
Phone: 740-355-9029

www.YeiTechnology.com
www.PrioVR.com
www.3SpaceSensor.com